

IMPROVED EVENT DATABASE MANAGEMENT METHOD AND SYSTEM FOR NETWORK EVENT REPORTING SYSTEM

COPYRIGHT NOTICE

5 A portion of the disclosure of this patent document contains material that is
subject to copyright protection. The copyright owner has no objection to the facsimile
reproduction by anyone of the patent document or the patent disclosure, as it appears in the
Patent and Trademark Office patent files or records, but otherwise reserves all copyright rights
whatsoever.

CROSS REFERENCE TO RELATED APPLICATIONS

10 This application is related to U.S. patent application serial no. 09/877,619, filed
June 8, 2001 and entitled "METHOD AND SYSTEM FOR EFFICIENT DISTRIBUTION OF
NETWORK EVENT DATA," attorney docket no. 3882/9, which application is hereby
incorporated herein by reference in its entirety.

BACKGROUND OF THE INVENTION

15 The invention disclosed herein relates generally to network monitoring systems.
More particularly, the present invention relates to improved methods and systems for efficiently
storing event data in a database and distributing the event data to different users, where the event
data relates to events occurring on a computer network.

20 Maintaining the proper operation of services provided over a network is usually
an important but difficult task. Service administrators are often called upon to react to a service
failure by identifying the problem that caused the failure and then taking steps to correct the
problem. The expense of service downtime, the limited supply of network engineers, and the
competitive nature of today's marketplace have forced service providers to rely more and more
25 heavily of software tools to keep their networks operating at peak efficiency and to deliver

contracted service levels to an expanding customer base. Accordingly, it has become vital that these software tools be able to manage and monitor a network as efficiently as possible.

A number of tools are available to assist administrators in completing these tasks.

One example is the NETCOOL® suite of applications available from Micromuse Inc. of San

5 Francisco, California which allows network administrators to monitor activity on networks such as wired and wireless voice communication networks, intranets, wide area networks, or the Internet. The NETCOOL® suite includes probes and monitors which log and collect network event data, including network occurrences such as alerts, alarms, or other faults, and store the event data in a database on a server. The system then reports the event data to network
10 administrators in graphical and text based formats in accordance with particular requests made by the administrators. Administrators are thus able to observe desired network events on a real-time basis and respond to them more quickly. The NETCOOL® software allows administrators to request event data summarized according to a desired metric or formula, and further allows administrators to select filters in order to custom design their own service views and service
15 reports.

In a demanding environment, there are many tens or even hundreds of clients viewing essentially the same filtered or summarized event data. Moreover, in a large network there are thousands of devices being monitored at a number of geographically distant locations, and events occur on these devices with great frequency. As a result, the databases that store
20 event data have become very large and are constantly being updated. Newly incoming event data is delayed before being stored or processed at the database, e.g., during a period when the databases are locked. Even if such delays are for a fraction of a second or a few seconds, this may impair the ability of the administrator clients to receive event data in a timely manner.

These and related issues become exacerbated as the size of a network increases, thus limiting the scalability of the network management system.

Accordingly, there is a need for improvements in how such network event databases are updated with events and how they are managed to provide greater scalability and efficiency. Furthermore, there is a need for improved techniques for efficiently coordinating the processing of event data obtained from both local and remote networks.

SUMMARY OF THE INVENTION

The present invention provides improved methods and systems for managing an event database in a network monitoring system. The improvements increase the efficiency in the way event data is handled by the database, increase the power and speed of the event database, and improve the scalability of the event database to allow it to serve a larger network. The improvements include methods for allowing users to set triggers to automatically pre-process event data received from monitored sites, a distributed system of local master and replica databases and methods for coordinating event data across them, and a notification subsystem for serving repeated client requests for raw and processed event data. These improvements work together to achieve improved performance as described herein.

Using the first aspect of the improvements, users are provided with the ability to add triggers into the event database. The triggers automatically test for certain event conditions, times, or database events, including system events or user-defined events including a name and selected parameter, and initiate a programmed action at the detection of such condition, time or event. One such trigger is a pre-processing trigger which examines event data before it is added to the event database, while still stored, e.g., in an event buffer or queue. One possible use of such a trigger is to prevent or reduce duplication of event data in the event database. That is, the

same or duplicated event data may be reported from one or more monitors, and the pre-processing of the event data detects the occurrence of duplicate data through comparison to event data already stored in the event database or other event data also received for but prior to insertion in the event database. Preventing duplication keeps the event database streamlined and also limits the time in which the event database is in a lock down condition such as may be necessary during read/write operations.

Moreover, the invention efficiently provides event data from both local and remote monitoring locations using a system of local replicas and their unioning. Each monitoring location may maintain its own event data of local monitored sites, as well as a replica of event data stored at other, remote monitoring locations. Using a union or combining operating, the local and remote data can be combined to provide a unified event data summary for use by a client. The monitoring locations update one another when their event data changes.

The combination of the pre-processing triggers and the use of replicas and unioning provides substantially improved scalability of the network monitoring system. Since data in local event databases is propagated to corresponding replicas in remote locations, it is particularly advantageous to avoid or delay updating each local database table unless desired or necessary. The use of triggers helps achieve this selection before tying up each database.

The improved scalability and power of the event database is further achieved through a publish/subscribe notification method and software component. The notification component registers persistent client requests for filtered raw data or summary data, associates similar requests with one another, and delivers all corresponding requests at about the same time to all users who request the same or similar data. This allows for users to be efficiently updated on new event data without excessively tying up the event database.

A common thread in the use of triggers, especially pre-processing triggers, a union/replica architecture, and a publish/subscribe notification system is that they are all event-based methodologies. Triggers are activated by the occurrence of events, be they temporal or otherwise, and initiate actions on the events themselves, e.g., refusal to insert, or immediate communication to a client. The union/replica architecture moves away from a traditional database model and toward a model which focuses on where events are occurring and what locations they might affect. The notification model focuses on increased efficiency in delivering specific event data or event data metrics to clients requesting it. Shifting the focus of a network management system to the events occurring on the network results in the improvements in speed and efficiency described herein.

BRIEF DESCRIPTION OF THE DRAWINGS

The invention is illustrated in the figures of the accompanying drawings which are meant to be exemplary and not limiting, in which like references are intended to refer to like or corresponding elements, and in which:

Fig. 1 is a block diagram showing functional components of an improved network monitoring system in accordance with one embodiment of the present invention;

Fig. 2 is a flow diagram showing the pre-processing of event data in accordance with one embodiment of the present invention;

Fig. 3 is a flow chart showing a process of using a trigger to pre-process event data in accordance with one embodiment of the present invention;

Fig. 4 is a flow diagram showing a process of managing local and replicas of event databases in accordance with one embodiment of the present invention;

Fig. 5 illustrates a client display with an ordered view in accordance with one

embodiment of the present invention;

Fig. 6 illustrates a client display with a map/geographical view in accordance with one embodiment of the present invention; and

Fig. 7 illustrates a client display with a list view in accordance with one
 5 embodiment of the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

In accordance with the invention, methods and systems are described herein, with reference to the Figures, for providing pre-processing of event data, and efficient delivery of the event data to a number of clients. In particular, the description herein focuses on a network
 10 monitoring system in which data is captured relating to events such as faults or alarms occurring on a computer network and is distributed to a number of administrator clients responsible for monitoring the network and preventing or correcting such faults.

Fig. 1 illustrates a network monitoring system in accordance with the present invention. The system includes an object server 26 which receives event data from a number of
 15 monitoring devices including probes 2 and monitors 4, stores the event data in one or more event databases 28, and provides the event data to a number of clients 8 which issue requests for the data. In one embodiment, the event databases 28 in the object server 26 are relational event databases 28 such as the Object Server database available as part of the NETCOOL®/Omnibus system available from Micromuse, Inc., San Francisco, California. Alternatively, the event
 20 database 28 may be any other suitable type of data store, such as an object-oriented database, flat file, etc. The event database 28 is a memory resident database, which is periodically dumped to file in case of failure. Events come in from probes 2 and monitors 4 in the form of SQL inserts. Clients 8 also access the database using SQL. As explained further below, the server is easy to

configure, and has increased performance, functionality and flexibility.

The probes 2 are portions of code that collect events from network management data sources 6, APIs, databases, network devices 5, log files, and other utilities. Monitors 4 are software applications that simulate network users to determine response times and availability of services 7 such as on a network. Other monitoring devices may be used to collect and report on events occurring in the network or related devices or services.

The network management system monitors and reports on activity on a computer, telecommunications, or other type of network. In this context, clients 8 are typically administrators who make requests for event data which they need to monitor on a regular basis. Clients may elect to see all event activity on the network. More typically for larger networks, clients will only want to see event data occurring on particular parts of the network for which they are responsible or which may affect their portion of the network. In addition, clients may only want to see summaries of their relevant part of the event data, such as event counts, sums, averages, *minimums*, *maximums*, or other distributions of event data. Clients input the various requests into an event list 34, with each request representing and being sometimes referred to herein as a particular view on the data.

Event data is stored in the event database 28 of one embodiment in a number of rows and columns, with each row representing an event and the columns storing fields of data relating to the event, e.g., location, type, time, severity, etc. As used herein, then, a view is generally a mechanism for selecting columns from the database and may also optionally include a filter. A filter is generally a mechanism for excluding rows of data in the database based on column values. Views may therefore be based on filters. Filters may also be based on other filters and other views. A metric view is generally a type of view that provides summary

information on the number of rows in a view rather than the actual data, and usually requires some arithmetic processing on the number of rows.

These client requests or views are persistent and are delivered according to a publish/subscribe model. That is, because network events occur regularly, the data in the event database 28 changes frequently and clients must be informed promptly of the updates in accordance with their specified requests to be able to make proper use of the data. The object server 26 processes the standing requests at a set frequency, e.g., every five or ten seconds, and delivers the results to the clients in the form of a stream or event data which is new or updated since the requests were last processed. The default or initial frequency for processing standing requests may be preset to any desired time frequency in any desired time units, e.g., seconds or portions thereof, minutes, hours, etc., and in any desired amount.

Regarding communications used at the object server 26, features of the present invention include event list optimization, replicas 42 and unions 44, and clusters. For event list optimization, a typical installation has many users with the same event lists. This is a significant cause of load at the server. Approaches include evaluating a view once, then publishing the results on the bus, and using throttling to prevent overload. A replica comprises a cached copy of a remote table. Inserts, updates, and deletes are passed to the master copy of a replica. A union is a set of tables and replicas, which are presented as a single table. Replicas and unions are discussed further below. Clusters are described in the above referenced commonly owned pending application serial no. 09/877,619, attorney docket number 3882/9.

In accordance with one aspect of the invention, a notification program or notifier 30 is provided which manages the client requests for data from the object server 26 to efficiently distribute the responses to the client requests. The notification program 30 may be part of the

object server 26 as shown or may be a separate, standalone component of the system. In accordance with processes described in greater detail in the above referenced commonly owned pending application, serial no. 09/877,619, attorney docket number 3882/9, the notification program 30 manages the various client requests in a view list or table 32 having a number of request sets. Each request set relates to a specific type of view or data filter and may include a number of metrics or formulas which summarize data in the object server 26 and which are requested to be processed by or for clients 8.

When the notifier 30 receives a registration request from a client 8, it scans its list of existing registrations. If, as in this example, an identical registration already exists, the second registration is associated with the first. The first registration of particular summary data may be referred to as a “primary” registration or request, whereas subsequent registrations of identical summary data may be referred to as “secondary” registrations or requests. The notifier 30 periodically scans its list of primary registrations, and for each it calculates the summary data, and sends the results to all clients that have registered interest in that data. Thus, when a client 18 elects a metric view in its event list 34, the notifier 30 registers interest in that metric view with the view table 32 in the object server 26. If another client elects to view the same metric view, the notifier 30 also registers that other client’s interest in the summary data in the view table 32.

As a result, this notification program 30 and view list library 32 optimizes the evaluation of summary data. Specifically, assuming that each client requests views of the same M metrics, the work done by the object server 26 is of the order of M, rather than $M \times (\text{number of clients})$.

Further in accordance with the invention, an automation engine 20 is provided as

part of the object server 26. The automation engine 20 involves the use of “triggers” to allow an administrator to connect events, conditions and actions together. The object server 26 provides the automation module with events, triggers and actions that are fully general to support many enhancement requests as well as debugging support wherein individual triggers can be put into
 5 debug mode, and debugging entries are written to a log file.

Much of the power of the object server 26 comes from automation. Automation allows the system to respond to events, i.e., happenings of interest such as the modification of data in a table, execution of a command, or the passing of a time interval. When an event occurs, a query is executed, and then a condition is evaluated, which if true causes the execution of an
 10 action: e.g., a sequence of SQL statements, an external script, or both. Moreover, each trigger has a coupling mode that indicates when the action should be executed, i.e., now or at some later time. Triggers may be created, altered, or dropped. The SQL command syntax for implementing triggers in one embodiment of the invention is summarized in the Appendix, which forms a part hereof.

Triggers allow actions to be executed when some event of interest occurs. The triggering event may be a primitive event, a database event, or a temporal event, for example. When the event occurs, an optional evaluate clause is executed, and then a condition is evaluated, and, if true, the action is executed. A primitive event is some happening of interest assigned a unique name. On occurrence, all events carry a set of arguments, which includes zero
 15 or more <name, type> pairs. The classes of primitive events include a user class for an event created by a user and a system class for some interesting occurrence in the object server, e.g. a user has logged in, the license is about to expire, etc.

Regarding user events, these may include user-defined events. Moreover, a user

event may be created at a command line: e.g. create event bob (a INT, b REAL). A user event may be raised at the command line: e.g. raise event bob 1, 1.2.

Temporal triggers signal an event at a determined frequency from a specified absolute time until a specified absolute time.

5 In accordance with an aspect of the invention, triggers may be inserted into the database to handle processing of event data before it gets added to the database. The object server receives raw event data from monitored sites in the network, e.g., via monitoring devices 2, 4. The raw event data is buffered in buffer 12, and pre-processed by the automations engine 20 prior to being stored in a database. Fig. 2 shows this process schematically, wherein trigger 10 22 reads event data in buffer 12 before it is inserted into event database table 40.

Advantageously, the event data 12 can be processed even while the database is in a read/write lockdown condition, e.g., when the database is being updated and cannot perform read or write operation from or to external devices. This avoids delays in processing the event data and communicating corresponding messages to the clients via a notifier.

15 An EECA model is used for triggers: event, evaluate, condition, action. When an event occurs, execute evaluate, then test the condition, and if true execute action. Transition tables may be used to communicate results between phases of a trigger. Procedural SQL syntax in actions may be used. Following are several exemplary types of triggers which may be used.

An “up-down correlation” temporal trigger may be described as follows:

20 NAME up-down correlation
 EVENT every 10 seconds
 EVALUATE select Node from alerts.status where ... bind as tt
 CONDITION when %row_count > 0

ACTION

for each row r in tt

update alerts.status set ... where Node = r.Node

5 A "PageOnRouterDown" temporal trigger may be described as follows.

NAME PageOnRouterDown

EVENT every 10 seconds

EVALUATE select * from alerts.status where... bind as tt

CONDITION true

10 ACTION

if %trigger.row_count > 0 AND (%trigger.positive_row_count mod 5) = 0

system(/usr/bin/page_message 'some routers are down')

end if

if %trgger.row_count = 0 AND (%trigger.zero_row_count mod 5) = 0

15 system(/usr/bin/page_message 'no routers are down')

end if

An example database trigger which preprocesses new event data to avoid duplication of event data in the event database is described as follows.

20 NAME deduplication

EVENT before reinsert on alerts.status

EVALUATE <nothing>

CONDITION true

ACTION

```

set old.Tally = old.Tally + 1,

set old.Summary = new.Summary

set old.LastOccurrence = new.LastOccurrence

```

5

An example event trigger may be described as follows.

NAME	ConnectionWatch
EVENT	on event connect
EVALUATE	<nothing>
10 CONDITION	true
ACTION	

```

insert into alerts.status... %event.user, %event,time

```

Referring to Fig. 3, a process of using triggers to preprocess event data such as in a deduplication trigger is as follows. A user generates the trigger based on a desired event and inserts it into the database, step 50. When new event data is received, step 52, it is checked by the trigger while still stored in the event buffer and before added to the database, step 54. If the event defined by the trigger has occurred, step 56, e.g., is represented by the event in the new event data, the trigger test to see if any user-defined condition(s) is satisfied, step 58. If so, the new event data is processed in accordance with the action specified in the trigger, step 60. For this purpose, the trigger generates a transition table, i.e., an anonymous table used by the action in the trigger, and uses one or more transition variables, i.e., variables generated by a database trigger and used by an action. If the action in the trigger involves deleting the new event data or otherwise avoiding or delaying insertion of the new event data into the event database, step 62,

the new event data is not added. Otherwise, or when the trigger action is not invoked, the event data is added to the event database, step 64.

Referring again to Fig. 1, in accordance with another aspect of the present invention, the event database 28 is distributed into local event databases 40 storing event data generated by a given location on the network and one or more replica event databases 42 storing event data gathered at and replicated from one or more remote network locations. When delivering on client requests, a union 44 is generated by combining the local and remote databases 40, 42, using commonly known unioning techniques.

For example, referring to Fig. 4, a distributed implementation is shown with one object server in London 26b, one in Hong Kong 26c, and another in New York 26a. Each object server 26a, 26b, 26c is configured to have a single table 40a, 40b, 40c, respectively, that holds locally generated alarms, and a replica of the table maintained by the other object server. So, for example, the New York local database table 40a holds locally generated alarms or other events in a table called alerts.NY, and has replicas of remotely generated alarms or other events in alerts.LN 42a and alerts.HK 43a. In addition it has a union 44a called alerts.status which logically contains alerts.HK, alerts.LN, and alerts.NY. A modification made to a local table is applied directly by that table. A modification to a replica normally requires the modification to be written through to its owner, e.g., an update of every row in alerts.status in Hong Kong results in a message being sent to New York requesting that the rows of alerts.NY held there be updated.

However, it is possible for a local monitoring location to take ownership of a replica, so that subsequent modifications to it do not require "writing through" to its owner that is, without receiving instructions from the remote monitoring location associated therewith. This

allows "follow-the-sun " configurations to be built, in which each local monitoring station controls updating of local and replica tables during its primary time period of operation, and then passes control at the end of this period to another location in a different time period which then assumes control for its given time period of business operations.

5 Thus, in accordance with the invention, a replica represents a copy of a table held on a remote object server and is incrementally updated to reflect the state of the table at the owning site. Attempts to modify data held in a replica result in a command being sent to the master copy of the table requesting the modification. A site holding a replica can become the owner if necessary. The attributes of a replica include its unique name in each database and its storage class (persistent or temporary). Attempts to modify data held in a union result in the command being applied to each of its components in an implementation-defined order. The attributes of a union include its unique name in each database and its storage class (persistent or temporary). Unions are created through specification of the names of the tables and replicas, and may be altered through the addition or removal of a table or replica.

10 The features described herein of the present invention support a powerful, scalable, and fast system for delivering network event data to clients using many different client views. Three exemplary views are shown in Figs. 5-7. Fig. 5 illustrates a client display with an ordered view in accordance with one embodiment of the present invention. The display is generated at the client location in response to the event data communicated to it by the server.

15 Both pure and metric views are shown (referred to as monitors in the screen displays, which refer to client-generated monitors and not specific monitor devices such as monitors or probes which collect data from specific networks, services, or devices), and the user can reorder views by drag and drop. Links appear as views. Moreover, a background image / pattern can be added.

Fig. 6 illustrates a client display with a map/geographical view in accordance with one embodiment of the present invention. The display may also be generated at the client location in response to the event data communicated to it by the server, and provide event information arranged geographically. Users can reposition views by drag and drop, resize views by dragging edges, add a link by selecting monitors to link, or add a background map. New monitors created in ordered view appear in unplaced palette.

Fig. 7 illustrates a client display with a list view in accordance with one embodiment of the present invention. This view provides event data based on the monitor. In particular, monitor information is displayed as a detailed list. One can sort on each column by selecting the column header.

Generally, the event data that is communicated to the client location may be presented in a variety of forms to suit the clients' needs.

Accordingly, it can be seen that the present invention provides improved and more efficient techniques for reducing the amount of work that needs to be performed by a database in a computer network in order to distribute event summary data to a large number of administrator clients. Moreover, the invention avoids or reduces delays experienced by event data at a database, e.g., due to delays in accessing a database. Furthermore, event data obtained from both local and remote networks is efficiently coordinated using replica and union processes. Each monitoring location in the network includes both locally generated events, and a copy of remotely-generated events which are provided and maintained by one or more remote monitoring locations.

While the invention has been described and illustrated in connection with preferred embodiments, many variations and modifications as will be evident to those skilled in

this art may be made without departing from the spirit and scope of the invention, and the invention is thus not to be limited to the precise details of methodology or construction set forth above as such variations and modification are intended to be included within the scope of the invention.

APPENDIX

SQL COMMAND SYNTAX SUMMARY

Triggers

CREATE [OR REPLACE] TRIGGER <name> // temporal triggers

[DEBUGGING <bool>]

[ENABLED <bool>]

PRIORITY <int> // 1=min, 20=max

[COMMENT <text>]

[FROM <abs_time>] [UNTIL <abs_time>] EVERY <rel_time>

[EVALUATE <select> BIND AS <name>]

[WHEN <condition>]

EXECUTE IMMEDIATE|DEFERRED|DETACHED

DECLARE

<decls>

BEGIN

<action>

END

CREATE [OR REPLACE] TRIGGER <name> // database triggers

[DEBUGGING <bool>]

[ENABLED <bool>]

PRIORITY <int> // 1=min, 20=max

[COMMENT <text>]

BEFORE|AFTER

INSERT|UPDATE|DELETE|REINSERT

ON <table>

FOR EACH { ROW | STATEMENT }

[EVALUTATE <select> BIND AS <name>] // for STATEMENT triggers

```

[WHEN <condition>]
EXECUTE IMMEDIATE|DEFERRED|DETACHED
DECLARE                                // pre-triggers must be immediate
    <decls>
5   BEGIN
    <action>
END

```

```

10  CREATE [OR REPLACE] TRIGGER <name>    // event triggers
    [DEBUGGING <bool>]
    [ENABLED <bool>]
    PRIORITY <int>                        // 1=min, 20=max
    [COMMENT <text>]
15  ON EVENT <name>
    [EVALUATE <select> BIND AS <name>]
    [WHEN <condition>]
EXECUTE IMMEDIATE|DEFERRED|DETACHED
    DECLARE
    <decls>
20  BEGIN
    <action>
END

```

```

25  ALTER TRIGGER <name>

SET DEBUG <boolval>

SET ENABLED <boolval>

SET PRIORITY <int>

```

DROP TRIGGER <name>

Trigger attributes

These are read-only scalar values accessible in WHEN and action blocks

5	%trigger.row_count	number of rows matched in evaluate clause
	%trigger.last_condition	value of condition on last execution
	%trigger.num_executions	number of times trigger has been run
	%trigger.num_fires	number of executions where condition was true
	%trigger.num_zero_row_count	number of consecutive fires with zero matches in eval
10	%trigger.num_postive_row_count	number of consecutive fires with >0 matches in eval